Europäisches Patentamt
European Patent Office
Office européen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

03290688.5

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 03290688.5
Demande no:

Anmeldetag:
Date of filing: 18.03.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

SCHLUMBERGER Systèmes
BP 620, 50 Avenue Jean Jaurès
92120 Montrouge
FRANCE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Procédé de sécurisation d'un ensemble électronique executant un algorithme quelconque contre les attaques par introduction d'erreur(s)

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F11/00

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL
PT SE SI SK TR LI

PROCEDE DE SECURISATION D'UN ENSEMBLE ELECTRONIQUE EXECUTANT UN ALGORITHME QUELCONQUE CONTRE LES ATTAQUES PAR INTRODUCTION D'ERREUR(S)

La présente invention concerne un procédé de sécurisation d'un ensemble électronique mettant en œuvre un algorithme quelconque où il est nécessaire de vérifier que le déroulement de l'algorithme a été effectué sans erreurs, que ce soit pour les étapes intermédiaires des différentes fonctions (déroulement du code, boucles, tests ...) ou pour les appels entre fonctions. Plus précisément, le procédé vise à réaliser une version de l'implémentation qui ne soit pas vulnérable face à un certain type d'attaques par introduction d'une ou plusieurs erreurs – dites « *Fault Analysis* » ou « *Extended Fault Analysis* » - qui cherchent à obtenir des informations sur une ou plusieurs données ou opérations intervenant dans le calcul à partir de l'étude du déroulement des calculs de l'ensemble électronique lorsqu'une ou plusieurs erreurs sont introduites.

La présente invention couvre toutes les implémentations où il est nécessaire de s'assurer que toutes les étapes intervenant lors du calcul ont été effectuées sans erreurs.

La première apparition de telles attaques date de 1996 :

- Ref (0): New Threat Model Breaks Crypto Codes – D. Boneh, R. DeMillo, R. Lipton – Bellcore Press Release

Le procédé vise également à proposer une parade aux attaques par rayonnement, flash, lumière, laser, glitch ou autres ou plus généralement à toute attaque perturbant l'exécution des instructions du programme dite attaque par perturbation. De telles attaques ont pour conséquence de modifier des instructions à exécuter et pour résultat l'inexécution ou la mauvaise exécution de certaines parties du programme.

Le procédé, objet de la présente invention, a pour objet la suppression des risques d'attaques par perturbation ou d'attaques par injection de faute d'ensembles ou systèmes électroniques en modifiant les fonctions qui interviennent lors du calcul.

Le procédé de sécurisation d'un ensemble électronique mettant en œuvre un processus de calcul classique qui doit être sans erreur, objet de la présente invention, est remarquable en ce que l'on modifie les fonctions mises en œuvres dans le calcul en y ajoutant en plusieurs endroits et ce de manière automatique :

- Une allocation de mémoire dynamique
- De l'information statique.
- Des objets de contrôles appelés « Balises » et « Points de contrôle ».
- Des appels à des fonctions dites « de balises » et « de vérification d'historique» qui de plus utiliseront la mémoire dynamique et statique.

Le procédé utilise le graphe de flot de contrôle du programme à protéger afin de générer les informations statiques utilisées par les fonctions de vérification.

La balise est une information qui définit les caractéristiques du point de passage correspondant et/ou d'un ou plusieurs autres points de passage.

Selon une forme de réalisation de l'invention, la balise est un nombre entier localisant la balise dans le code à protéger.

Selon une autre forme de réalisation, la balise est un booléen définissant s'il s'agit de la première ou de la dernière balise.

Selon une forme particulière de l'invention, la balise est une structure de donnée caractérisant, en fonction de la valeur d'un registre ou d'une variable donnée, l'ensemble des balises dont on interdira (par une fonction de vérification) le passage dans la suite du déroulement de l'exécution.

Selon une autre forme de réalisation particulière de l'invention, la balise est une structure de donnée caractérisant, en fonction de la valeur d'un registre ou d'une variable donnée, l'ensemble des balises dont on forcera (par une fonction de vérification) le passage dans la suite du déroulement de l'exécution.

Les formes de réalisation citées peuvent être combinées : la balise peut être constituée de plusieurs des éléments décrits précédemment.

Les fonctions de balises et de vérifications utilisent une partie partagée de la mémoire du système.

La mémoire partagée contient une structure de donnée de type pile, de type tableau de booléens et/ou de type nombre stockant une somme de contrôle.

Une fonction de balise est une fonction qui est appelée par le programme à chaque passage par une balise et qui consiste à stocker dynamiquement dans la mémoire partagée diverses informations concernant la balise, et éventuellement à effectuer un contrôle sur le déroulement de l'exécution.

Selon une forme de réalisation particulière de l'invention, la fonction de balise est une fonction qui empile la balise dans la mémoire partagée.

Selon une autre forme de réalisation, la fonction de balise est une fonction qui met à jour une somme de contrôle contenue dans la mémoire partagée avec des données de la balise.

Lesdites formes de réalisation peuvent être combinées.

Une fonction de vérification d'historique est une fonction appelée à chaque point de contrôle afin de vérifier la cohérence des informations stockées dans la mémoire partagée lors des appels successifs des fonctions de balises.

La mémoire statique calculé par le processus automatique est une liste de pile de balises représentant un historique valide de passage de balises et/ou une liste de sommes de contrôle obtenues à partir d'un historique valide de passage de balises.

Dans le cas d'une somme de contrôle, la somme de contrôle est obtenu par l'utilisation de la fonction addition, par l'utilisation de la fonction Ou exclusif (ou XOR), par l'utilisation d'un checksum usuel et/ou par l'utilisation d'une fonction cryptographique de hachage.

Les objets « Balise » et « Point de contrôle », les fonctions « Fonction de Balises » et « Fonction de Vérification d'historique » ainsi que la mémoire partagée seront décrites de manière générique par la suite et des exemples seront également donnés.

DESCRIPTION SOMMAIRE DES DESSINS

D'autres buts, avantages et caractéristiques de l'invention apparaîtront à la lecture de la description qui va suivre de la mise en oeuvre du procédé selon l'invention et d'un mode de réalisation d'un ensemble électronique adapté pour cette mise en œuvre, donnés à titre d'exemple non limitatif en référence aux dessins ci-annexés dans lesquels :

-la figure 1 montre une représentation schématique d'un mode de réalisation d'un module électronique selon la présente invention ;

-la figure 2 montre un graphe représentant des étapes du procédé selon la présente invention.

MANIERE DE REALISER L'INVENTION

Le procédé selon l'invention vise à sécuriser un ensemble électronique, et par exemple un système embarqué tel qu'une carte à puce mettant en œuvre un programme. L'ensemble électronique comprend au moins un processeur et une mémoire. Le programme à sécuriser est installé dans la mémoire, par exemple de type ROM dudit ensemble.

A titre d'exemple non limitatif, l'ensemble électronique décrit dans ce qui suit correspond à un système embarqué comprenant un module électronique 1 illustré sur la figure 1. De tels modules sont réalisés le plus souvent sous la forme d'un microcircuit électronique intégré monolithique, ou puce, qui une fois protégé physiquement par tout moyen connu peut être monté sur un objet portatif tel que par exemple une carte à puce, carte à microcircuit ou autre utilisable dans divers domaines.

Le module électronique 1 à microprocesseur comprend un microprocesseur CPU 3 relié de façon bidirectionnelle par un bus 5 interne à une mémoire 7 non volatile de type ROM, EEPROM, Flash, FeRam ou autre contenant le programme PROG 9 à exécuter, une mémoire 11 vive de type RAM, des moyens 13 I/O d'entrée/sortie pour communiquer avec l'extérieur.

1. PRINCIPE

1.1 Balises et fonction de balises

Définition

- Une balise sera une information qui définira les caractéristiques du point de passage.

- Une fonction de balises sera une fonction qui sera appelée par le programme à chaque passage par une balise et qui consistera à stocker dans la mémoire partagée diverses informations concernant la balise et éventuellement à effectuer un contrôle sur le déroulement de l'exécution.

Exemples

Une balise pourra par exemple être :

- Un nombre entier qui permet de connaître la localisation de la dites balise ou bien un booléen définissant s'il s'agit de la première ou de la dernière balise par exemple.
- Une structure complexe décrivant un ensemble d'information concernant l'état actuel du dispositif électronique exécutant le code. Par exemple, une structure de données caractérisant, en fonction de la valeur d'un registre ou d'une variable donnée, l'ensemble des balises dont on interdira (par la fonction de balise ou celle de vérification) le passage dans la suite du déroulement de l'exécution.

Une fonction de balises pourra par exemple effectuer les opérations suivantes :

- Regarder si la balise est la première balise (à l'aide d'un champ spécifique à la balise). Si oui créer une pile vide en mémoire partagée, empiler la balise et continuer le code, sinon empiler la balise dans la pile et continuer l'exécution du code.

1.2 Points de contrôle, Vérification d'historique

Définition

- Un point de contrôle sera une structure de donnée contenant des informations qui seront utilisées par la fonction de vérification d'historique.

- La fonction de vérification d'historique est une fonction appelée à chaque point de contrôle afin de vérifier la cohérence des informations stockées dans la mémoire partagée lors des appels successifs des fonctions de balises.

Exemples

- Un point de contrôle peut être constitué de toutes les listes de balises correspondant à des déroulements d'exécution autorisés aboutissant à ce point de contrôle.
- La fonction de vérification pourra être la vérification que la pile des balises traversées correspond à une des listes du point de contrôle. Si tel n'est pas le cas une erreur est détectée.

2. FORME DE REALISATION

Nous décrivons ici à titre d'exemple le fonctionnement d'un préprocesseur de code C qui implante la detection d'erreur de façon semi-automatique.

Le préprocesseur accepte comme entrée du code source C, et transforme ce code pour y inclure la détection d'erreur.

Cette transformation est guidée par des directives incluses dans le code qui spécifient les balises qui doivent être introduites. Ces directives peuvent prendre les formes suivantes:

1.start : cette directive spécifie que la pile des balises successivement traversées doit être vidée.

2.flag : cette directive spécifie l'introduction d'une nouvelle balise qui doit être empilée à chaque passage.

3.verify : cette directive spécifie l'introduction d'un point de contrôle qui provoquera, à chaque fois qu'il sera traversé, la vérification que l'ensemble des balises empilées correspond à un déroulement d'execution admissible.

4.race n cond : cette directive spécifie l'introduction d'une balise qui indique lorsqu'elle est traversée que si l'expression booléenne cond est vraie, alors seuls les déroulements d'execution de la famille n sont admissibles. Une telle famille est définie par l'ensemble des directives du type suivant.

5.flag !n1 ... !nk m1 ... mk : cette directive spécifie que les déroulements d'execution des familles n1...nk ne doivent pas passer par ce point, et que les déroulements d'execution des familles m1...mk doivent passer par ce point.

6.Loop n : cette directive spécifie l'entrée dans une boucle dont le nombre de tour est n.

On donne ici un un exmple en langage C comportant des directives à l'attention du préprocesseur. La fonction int f(int x, int d) effectue action1(d) trois fois, puis action2(d) si x est égal à 1. La fonction action2(int d) effectue action21(d) puis action22(d).

Les directives sont placées dans le code sous forme de #pragma. Leurs effets sont indiqués en commentaire.

```
int f (int x,int d) {
  int i;
  /* starting point of the program part to be protected */
#pragma start
  /* definition of the possible scenarios (optional) */
#pragma race 0 x != 1
#pragma race 1 x == 1
  for(i=0; i<3; i++) {
    /* tells cfprotect that this loop has 3 turns */
#pragma loop 3
#pragma flag
    action1(d);
  }
  /* all the race must pass here, the flags are automatically numbered */
```

```
#pragma flag

  if (x == 1) {
    action2(d);
  }

  /* verification of the stack consistency, i.e. that the stack of flags
     is consistent regarding the control flow of the program */
#pragma verify
}

void action2(int d) {
  /* the race 1 must pass here and race 0 must not */
#pragma flag !0 1
#pragma verify
  action21(d);
  action22(d);
}
```

Le code obtenu en sortie du précompilateur est le suivant:

```
/*****************************************************************/
#ifdef CONTROL_FLOW_PROTECTION
#ifndef CONTROL_FLOW_PROTECTION_HEADER
#define CONTROL_FLOW_PROTECTION_HEADER
#include <string.h>
char __control_flow_stack[8];
char __cf_stack_index=0;
char __cf_race_flags=0;
char __fcPath0[] = {2, 3, 3, 3, 4, 0};
char __fcPath1[] = {2, 3, 3, 3, 4, 1, 0};
#define __cfSET_RACE(x) __cf_race_flags |= x
#define __cfRACE(x) __cf_race_flags & x
#define __cfNORACE !(__cf_race_flags)
#define __cfPUSH(x) __control_flow_stack[__cf_stack_index]=x, __cf_stack_index++
#define __cfRESET(x) __control_flow_stack[0]=x,__cf_stack_index=1,__cf_race_flags=0
#define __cfVERIFY(p) strcmp(__control_flow_stack,p)==0
#define __cfERROR printf("control flow error detected\n")
#endif
#endif
/*****************************************************************/

int f (int x, int d) {

  int i;

  /* starting point of the program part to be protected */
#ifdef CONTROL_FLOW_PROTECTION
__cfRESET(2);
#endif

  /* definition of the possible scenarios (optional) */
#ifdef CONTROL_FLOW_PROTECTION
if (x != 1) __cfSET_RACE(1);
#endif
#ifdef CONTROL_FLOW_PROTECTION
if (x == 1) __cfSET_RACE(2);
#endif

  for(i=0; i<3; i++) {
    /* tells cfprotect that this loop has 3 turns */
#pragma loop 3
#ifdef CONTROL_FLOW_PROTECTION
__cfPUSH(3);
#endif
    action1(d);
  }

  /* all the race must pass here, the flags are automatically numbered */
#ifdef CONTROL_FLOW_PROTECTION
__cfPUSH(4);
#endif

  if (x == 1) {
    action2(d);
  }
```

```
  /* verification of the stack consistency, i.e. that the stack of flags
     is consistent regarding the control flow of the program */
#ifdef CONTROL_FLOW_PROTECTION
  __control_flow_stack[__cf_stack_index]=0;
  if (!((__cfNORACE && (__cfVERIFY(__fcPath0) || __cfVERIFY(__fcPath1))) ||
        ((!(__cfRACE(1)) || (__cfVERIFY(__fcPath0))) &&
         (!(__cfRACE(2)) || (__cfVERIFY(__fcPath1))))))
    { __cfERROR; }
#endif
}

void action2(int d) {
  /* the race 1 must pass here and race 0 must not */
#ifdef CONTROL_FLOW_PROTECTION
__cfPUSH(1);
#endif
#ifdef CONTROL_FLOW_PROTECTION
  __control_flow_stack[__cf_stack_index]=0;
 if (!((__cfNORACE && (__cfVERIFY(__fcPath1))) ||
        ((!(__cfRACE(1)) &&
         (!(__cfRACE(2)) || (__cfVERIFY(__fcPath1)))))))
    { __cfERROR; }
#endif
  action21(d);
  action22(d);
}
```

Le précompilateur définit des structures de données utilisées pour la vérification. Ces structures de données se divisent en deux types: statiques et dynamiques.

Les structures de données dynamiques sont:

1. Un tableau d'octets destiné à stocker la pile de balises successivement traversées au fur et à mesure de l'exécution du programme.
2. Un octet indiquant la taille de la pile.
3. Un octet désignant les familles de chemins admissibles.

```
char __control_flow_stack[8];
char __cf_stack_index=0;
char __cf_race_flags=0;
```

Les données statiques décrivent les piles de balises admissibles; dans notre cas:

```
char __fcPath0[] = {2, 3, 3, 3, 4, 0};
char __fcPath1[] = {2, 3, 3, 3, 4, 1, 0};
```

Ces données sont calculées par le préprocesseur à partir du graphes de flôt de contrôle du programme, calculé également par le préprocesseur (voir figure 2). L'analyse de ce graphe indique quelles sont les suites de balises correspondant effectivement à un déroulement d'execution menant d'une balise d'initialisation à une balise de vérification.

Le précompilateur définit également un ensemble de fonctions, en fait des macros C, utilisées lors les passages de balises. Ces fonctions sont les suivantes:

```
#define __cfSET_RACE(x) __cf_race_flags |= x
#define __cfRACE(x) __cf_race_flags & x
#define __cfNORACE !(__cf_race_flags)
#define __cfPUSH(x) __control_flow_stack[__cf_stack_index]=x, __cf_stack_index++
#define __cfRESET(x) __control_flow_stack[0]=x,__cf_stack_index=1,__cf_race_flags=0
#define __cfVERIFY(p) strcmp(__control_flow_stack,p)==0
#define __cfERROR printf("control flow error detected\n")
```

1. __cfSET_RACE(x) est utilisées pour indiquer que la famille de déroulement d'execution codée par x est admissible.
2. __cfRACE(x) teste si la famille codée par x a été effectivement activée, dans ce cas, seul les déroulement d'execution de cette famille sont admis.
3. __cfNORACE indique qu'aucune famille n'a été activée, dans ce cas, tout déroulement d'exécution cohérent avec le graphe de flôt de contrôle est admissible.
4. __cfPUSH(x) empile la balise x; le précompilateur assigne à chaque balise start ou flag un identificateur unique.
5. __cfRESET(x) vide la pile de balises.
6. __cfVERIFY(p)compare la pile de balises avec un tableau représentant une pile admissible (l'un des tableaux statiques précalculés).
7. __cfERROR inque la procedure à executer en cas de détection d'erreur.

Ces fonctions sont utilisées pour implanter la détection d'erreur:

- Une directive start est remplacée par une balise d'initialisation codée par __cfRESET(x) où x est le symbole attribué à la balise en question.
- Une directive flag ... est replacée par par une balise codée par __cfPUSH(x) où x est le symbole attribué à la balise en question.
- Une directive race n cond est remplacée par

  if (cond) __cfSET_RACE(x)

  où x code la famille n de déroulement d'execution.
- Une directive verify est remplacée par un test, spécifique au point de programme où elle figure, qui vérifie que la pile de balises correspond à un déroulement d'execution admissible, en tenant compte des familles activées.

Cette vérification s'appuie notamment sur les données statiques calculées par le préprocesseur.

- Les directives loop_n sont utilisées par le précompilateur uniquement qui les retire après sont calcul.

Dans le cas de l'exemple proposé, les vérifications s'effectuent à la fin de la fonction f et dans la fonction action2. Les tableaux __fcPath0 et __fcPath1 définissent les déroulement d'execution admissibles. Il y a deux familles d'execution la premier est constituée de __fcPath0 et la seconde de __fcPath1. __fcPath0 correspond à l'execution qui ne passe pas par l'appel de action2(d); __fcPath1 correspond à l'execution qui passe par l'appel de action2(d).

Si une vérification échoue, cela indique que la pile représentant les balises successivement traversées ne concorde pas avec le graphe de flôt de contrôle du programme. On en déduit qu'une erreur a été injectée lors de l'execution.

Le procédé de sécurisation est remarquable en ce que l'on modifie les fonctions mises en œuvres dans le calcul en y ajoutant en plusieurs endroits et ce de manière automatique :

1. Des informations statiques générées par le procédé automatique.
2. Une partie dynamique de la mémoire du système électronique allouée par le procédé automatique.
3. Des balises et des points de contrôles pour jalonner le code, introduits par le procédé automatique.
4. Des fonctions de balises stockant des informations dans la mémoire dynamique.
5. Des fonctions de vérification d'historique utilisant les informations statiques et la mémoire dynamique pour vérifier qu'aucune erreur n'a été introduite.

REVENDICATIONS

1- Procédé de sécurisation de l'exécution d'au moins un module dans une unité électronique comportant des moyens de traitement de l'information et des moyens de mémorisation d'informations caractérisé en ce que, durant l'exécution dudit module, il consiste, lors du passage par au moins une balise, à stocker une ou plusieurs informations sur une ou des caractéristiques de ladite balise et à vérifier, au moins en un point de contrôle, la consistance des informations stockées sur l'ensemble des balises rencontrées.

2- Procédé de sécurisation d'au moins un module destiné à être intégré dans une unité électronique comportant des moyens de traitement de l'information et des moyens de mémorisation d'informations caractérisé en ce qu'il intègre automatiquement audit module muni d'un ensemble de directives un ensemble de données statiques, de fonctions de balise et de fonctions de vérification, les premières représentant un ensemble de déroulements d'exécutions valides, les secondes calculant dynamiquement une représentation de l'exécution, et les dernières servant à vérifier la cohérence des données statiques et dynamiques.

3- Procédé selon la revendication 2, caractérisé en ce qu'il utilise le graphe de flot de contrôle du programme à protéger afin de générer les informations statiques utilisées par les fonctions de vérification.

4- Procédé selon la revendication 2 ou 3, caractérisé en ce qu'une balise est une information qui définit les caractéristiques du point de passage correspondant et/ou d'un ou plusieurs autres points de passage.

5- Procédé selon la revendication 4, caractérisé en ce qu'une balise est un des éléments suivants ou une combinaison de plusieurs ou de l'ensemble des éléments suivants :

- un nombre entier localisant la balise dans le code à protéger ;
- un booléen définissant s'il s'agit de la première ou de la dernière balise ;

- une structure de donnée caractérisant, en fonction de la valeur d'un registre ou d'une variable donnée, l'ensemble des balises dont on interdira (par une fonction de vérification) le passage dans la suite du déroulement de l'exécution ;
- une structure de donnée caractérisant, en fonction de la valeur d'un registre ou d'une variable donnée, l'ensemble des balises dont on forcera (par une fonction de vérification) le passage dans la suite du déroulement de l'exécution.

6- Procédé selon l'une des revendications 2 à 5, caractérisé en ce qu'une fonction de balise est une fonction qui sera appelée par le programme à chaque passage par une balise et qui consistera à stocker dynamiquement dans la mémoire partagée diverses informations concernant la balise.

7- Procédé selon la revendication 6, caractérisé en ce qu'une fonction de balise est une fonction qui empile la balise dans la mémoire partagée et/ou une fonction qui met à jour une somme de contrôle contenue dans la mémoire partagée avec des données de la balise.

8- Procédé selon l'une des revendications 2 à 7, caractérisé en ce qu'une fonction de vérification d'historique est une fonction appelée à chaque point de contrôle afin de vérifier la cohérence des informations stockées dans la mémoire partagée lors des appels successifs des fonctions de balises.

9- Unité électronique comportant des moyens de traitement de l'information et des moyens de mémorisation d'informations contenant au moins un module à exécuter caractérisé en ce qu'il comporte des moyens permettant durant l'exécution dudit module, et lors du passage par au moins une balise, à stocker une ou plusieurs informations sur une ou des caractéristiques de ladite balise dans lesdits moyens de mémorisation et des moyens permettant de vérifier, au moins en un point de contrôle, la consistance des informations stockées sur l'ensemble des balises rencontrées.

10 - Programme comprenant des instructions de code de programme pour l'exécution des étapes du procédé selon l'une des revendications 1 à 6 lorsque ledit programme est exécuté dans une unité électronique.

5

ABREGE DESCRIPTIF

L'invention concerne un procédé automatique de sécurisation d'un ensemble électronique de calcul contre les attaques par introduction d'erreur ou par rayonnement.

L'on utilise

1. Des informations statiques générées par le procédé automatique.

2. Une partie dynamique de la mémoire du système électronique allouée par le procédé automatique.

3. Des balises et des points de contrôles pour jalonner le code, introduits par le procédé automatique.

4.Des fonctions de balises stockant des informations dans la mémoire dynamique.

5.Des fonctions de vérification d'historique utilisant les informations statiques et la mémoire dynamique pour vérifier qu'aucune erreur n'a été introduite.

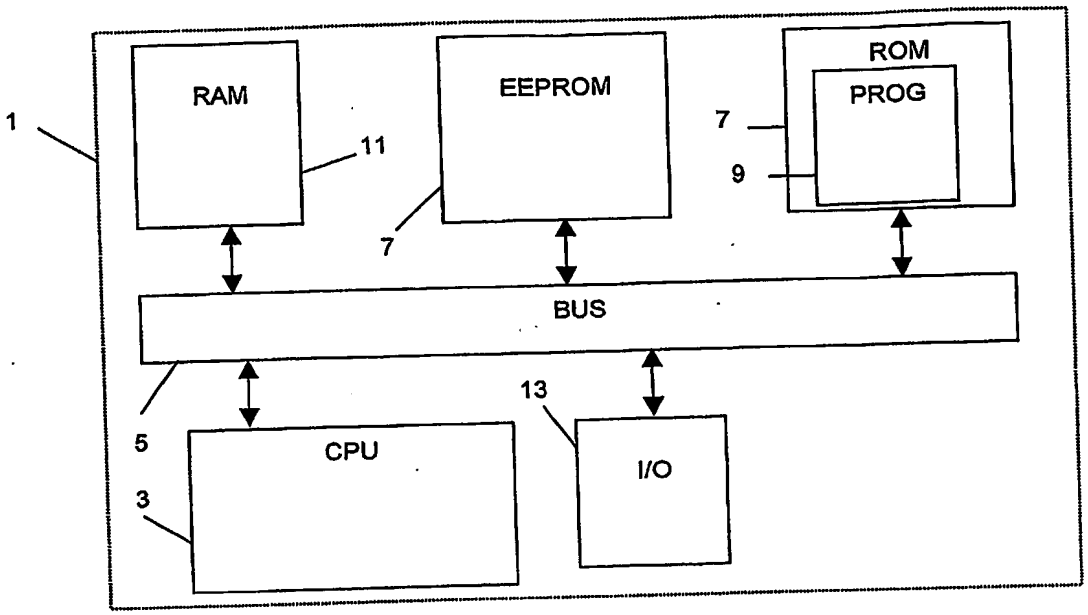

FIG.1

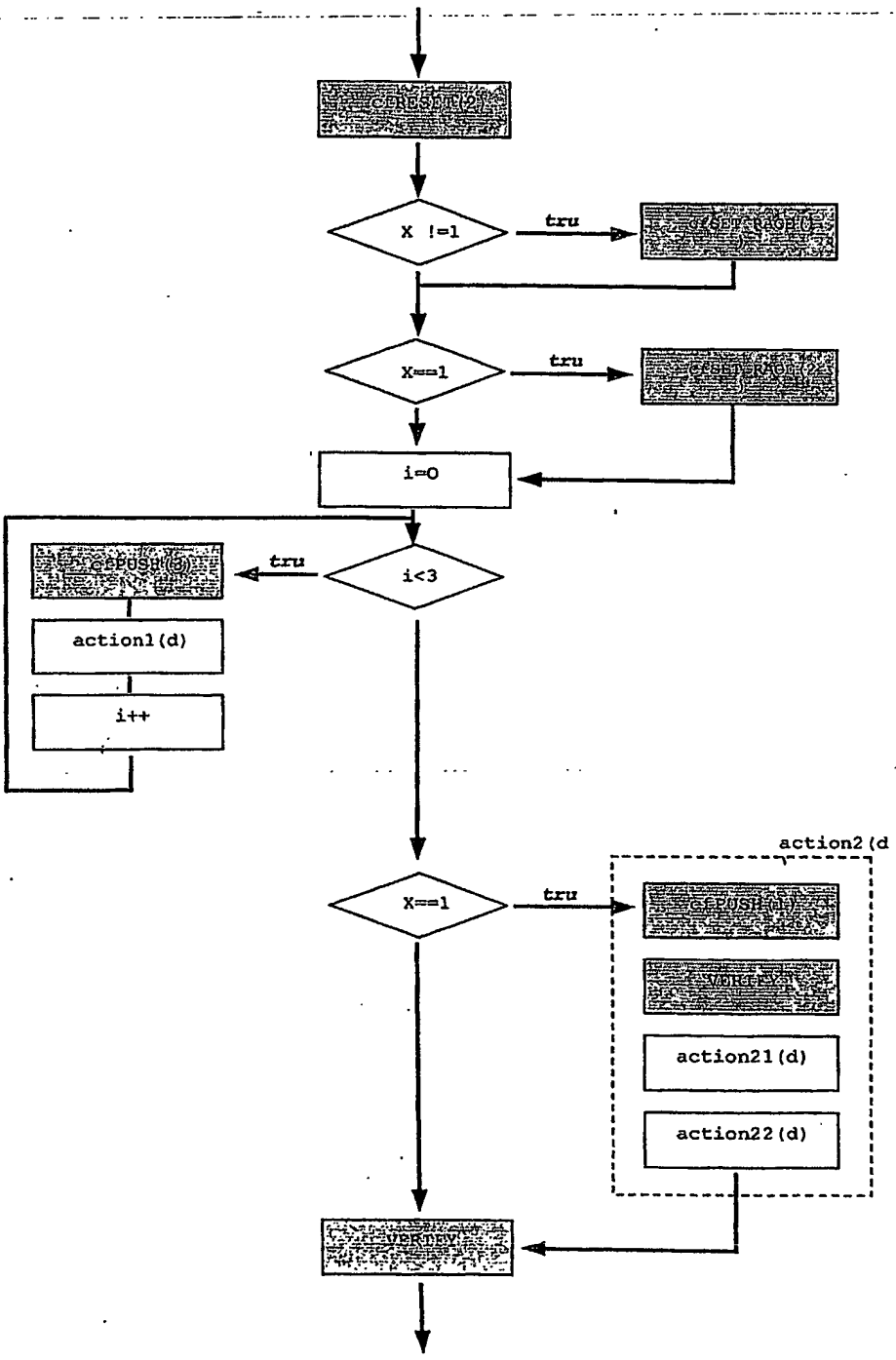
X !=1
tru
X==1
tru
i=0
i<3
tru
action1(d)
i++
action2(d
X==1
tru
action21(d)
action22(d)

FIG.2

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- [x] BLACK BORDERS
- [x] IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- [x] FADED TEXT OR DRAWING
- [ ] BLURED OR ILLEGIBLE TEXT OR DRAWING
- [ ] SKEWED/SLANTED IMAGES
- [x] COLORED OR BLACK AND WHITE PHOTOGRAPHS
- [ ] GRAY SCALE DOCUMENTS
- [ ] LINES OR MARKS ON ORIGINAL DOCUMENT
- [ ] REPERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- [ ] OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning documents *will not* correct images problems checked, please do not report the problems to the IFW Image Problem Mailbox